The .NET Framework Practices And Patterns

Brad Abrams
Program Manager
.NET Framework
Microsoft Corporation

3-327

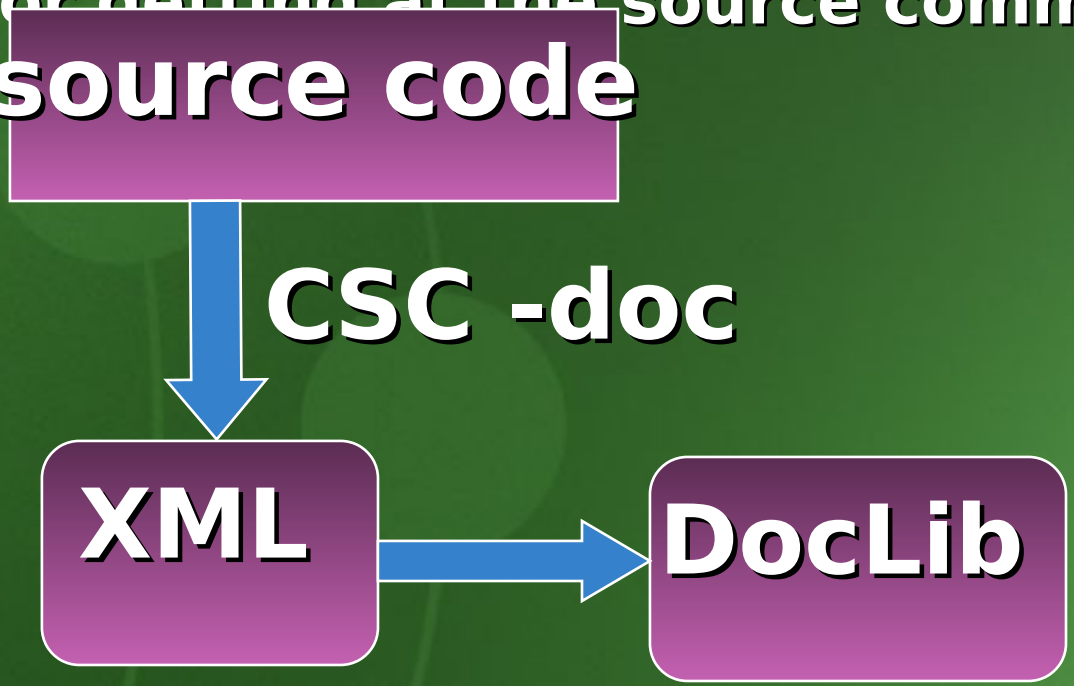

Microsoft®
PDC 2000
Professional Developers Conference
Microsoft®
.net™
the defining point

Introduction

- Examine some best practices for writing managed code
- See how the design practices are
  used in a sample component
- Understand how and when to apply these practices in your code

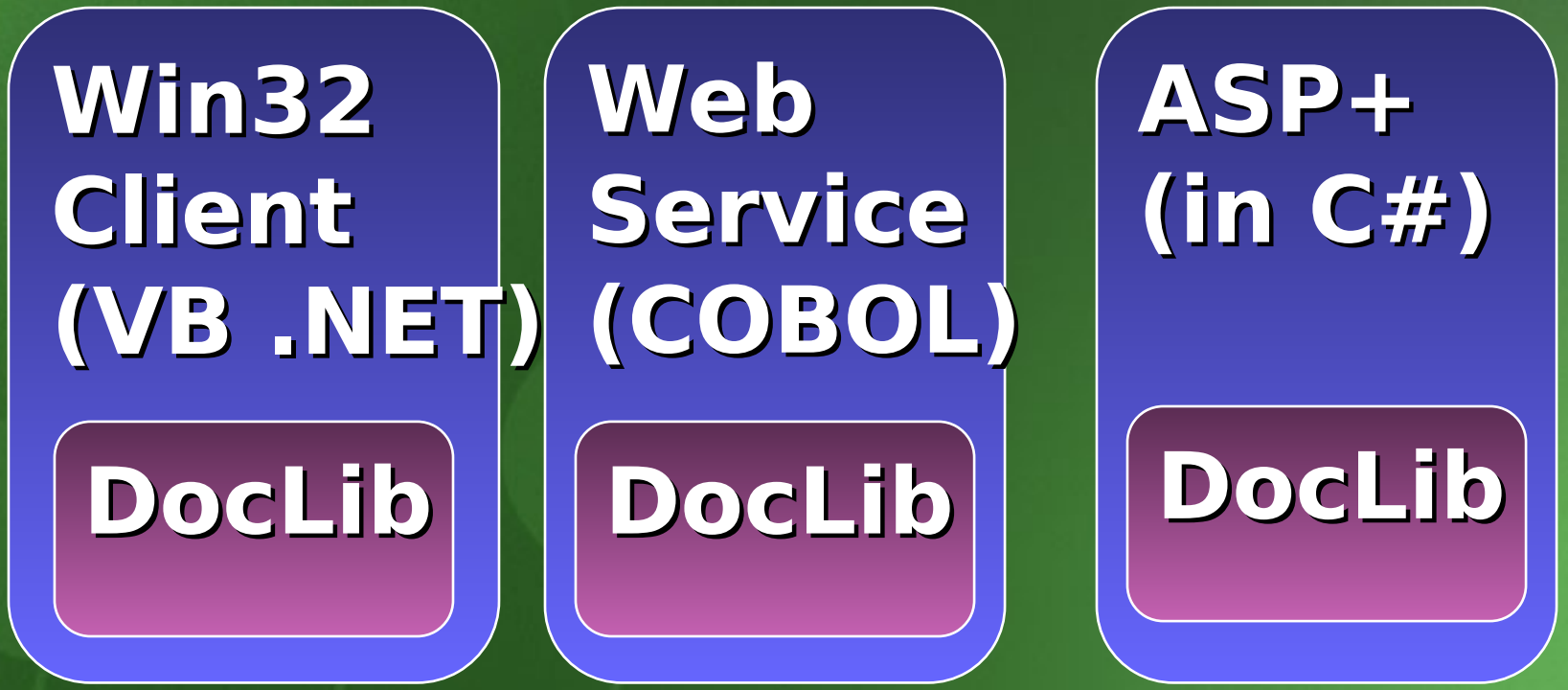
The Sample Component

- A shared library for accessing documentation
- In source comments are turned into XML
- DocLib parses the XML and provides an API for getting at the source comments



The Sample Component

- DocLib is used in several different applications



About Design Guidelines

- Design guidelines are by definition contentious
- Many have no “right” answer
- I’ll show what we decided to do and why
- There is leverage in using the same guidelines

Common Language Specification

- Guarantees interoperability between languages
- No need to pick a “target language”
- Rich coherent subset
- Compiler enforcement

Naming Patterns

- Identifiers should be readable
  - Avoid abbreviations
  - Don’t use underscores
  - Don’t use all caps
  - Don’t use Hungarian
- Instead of `lpcName` use `Name`
- Instead of `MAX_VALUE` use `MaxValue`

Naming Patterns

- PascalCasing and camelCasing

```
public class MemberDoc {
   int CompareTo (object value) {
      double currentVersion = 1.0
   }
   string Name {get;}
}
```

- Familiar to many users
- Used consistently in the frameworks

Error Handling

- Use typed exceptions rather than
  error codes
- Robust: failures get noticed
- Use only in exceptional situations, e.g.:
  - Reading byte-by-byte to the EOF is NOT an exception, it happens every time

Properties Versus Methods

- Do use a Property:
  - If the member has a logical data member

```
string Name {get;}//Property
Guid GetNext(){}  //Method
```

- Do use a Method:
  - If the operation is a conversion `ToString()`
  - If the getter has an observable side effect

Properties Versus Methods

- Do use a method: (continued)
  - If order of execution is important
  - If the member returns an array

```
EmployeeList l = FillList();
for (int i = 0; i < l.Length; i++){
    if (l.All[i] == x){...}
}
```

  - Creates a new snap shot of the array each time through the loop

Properties Versus Methods

- Do use a method: (continued)
  - If order of execution is important
  - If the member returns an array

```
EmployeeList l = FillList();
for (int i = 0; i < l.Length; i++){
    if (l.GetAll()[i]== x){...}
}
```

  - Creates a new snap shot of the array each time through the loop

Threading

- Frameworks are free threaded (not thread safe) by default
  - Avoids deadlocks and contention
  - In ASP+, user code runs in a single thread at a time
  - Allows locks to be taken at a coarse granularity
- Static methods must be thread safe
- Many classes are thread safe (ex. Console)

Threading

- Explicit pattern for getting thread safe instances
  - Immutable (ex. String) – objects that have no settable state are safe
  - ReadOnly (ex. DataSet) – Editable and unsafe until ReadOnly() is called and then it is immutable
  - Synchronized (ex. Collections) – Uses
    a synchorinzed wrapper

Memory Management

- Small, short lived objects are cheap
  - Stays in the L2 cache (on processor)
  - Allocates and collects an object in 45 clock cycles (that's ~10 million objects per sec)
- Create temp objects where it makes the programming model simpler

```
if (s.ToLower().Trim() == “foo”)
```

- All this is an optimization

Memory Management

- Non-deterministic lifetime
  - No guarantees on when the finalizer will run to clean up resources
  - Provide a Dispose or Close pattern to explicitly free resources
  - Only implement Finalize on a class that needs external resource cleanup

Reference Types And Value Types

- Reference Types
  - GCHeap allocated
  - Assignments copy the reference
  - Arrays allocated out of line
  - Passed by reference
- Value Types (structs):
  - Stack allocated
  - Assignments copy the value
  - Arrays allocated in line
  - Passed by value

Reference Types And Value Types

- Boxing
  - Changes a Value Type to a Reference Type by copying it onto the GCHeap
  - Expensive in a tight loop
  - Example: Strongly typed StringBuilder.Append overloads

Reference Types And Value Types

- Value Types
  - Value semantics (such as an int)
  - Small instance size (< 16 bytes)
- Use Reference Types everywhere else

Interfaces Versus Base Class

- Favor using Base Classes over Interfaces
  - Base Classes version better
  - Methods can be added with a default implementation
  - Can provide concrete implementation - ActiveX® versus Win Forms controls
- Use Interfaces when multiple roots are required
  - IServiceObjectProvider – Many different types
    of things can be service providers

Related Sessions And References

- Other PDC sessions recommended:
  - 3-313 An Architectural Overview of the .NET Framework
  - 5-313 Introduction to C#
  - 3-222 Handling the Basics: Getting the Most Out of the .NET Framework
- Other reference materials recommended:
  - Class Library Design Guidelines
  - Technical Overview of the Common Language Runtime

Questions?

Where do you want to go today?

Microsoft